# CAKE-HI – Compact Authenticated Key Exchange Hiding Identities

Uri Blumenthal*, Gene Itkis*, Roger Khazan*, Brandon Luo*, Sean O'Melia*, Brian Proulx*, David Stott*, Gabriel Torres*, David A. Wilson*

* *Massachusetts Institute of Technology Lincoln Laboratory, Lexington, MA, USA*

*Email: {uri,itkis,rkh,Brandon.Luo,Sean.Omelia,Brian.Proulx,David.Stott,Gabriel.Torres,David.Wilson}@ll.mit.edu*

***Abstract*—Modern public-key cryptography is threatened by advances in quantum computing. As a result, there has been a shift towards cryptographic algorithms that can resist attacks by a quantum computer. However, these algorithms use significantly longer keys, and produce larger ciphertexts and digital signatures than their classical counterparts. These bigger sizes pose problems for devices that are bandwidth- and/or power-limited, and wish to establish a secure, quantum resistant communication channel with another device.**

**In order to reduce the overhead of using these algorithms in challenging environments while maintaining security posture, we present Compact Authenticated Key Exchange – Hiding Identities (CAKE-HI). To evaluate our protocol, we compare the key exchange handshake size and computational efficiency of mutual authenticated TLS and CAKE-HI. Measurements show that CAKE-HI significantly reduces the handshake size and the computational overhead of establishing a quantum-secure link.**

**In addition, we formalize and prove security properties about CAKE-HI in the symbolic and computational model using the protocol analysis frameworks Verifpal and CryptoVerif.**

***Index Terms*—authenticated, compact, cryptography, key exchange, quantum-resilient, bandwidth-limited, power-limited, minimalistic**

## 1. Introduction

A cryptographic *key exchange* is the process by which two parties can mutually agree on a shared secret key, even in the presence of an eavesdropping adversary. The key exchange process is the first step in establishing secure communication between the two parties. For efficiency, they must share a *symmetric key* – a short (e.g., 256-bit) random string that is known only to these two parties. The symmetric key is a product of the key exchange, either directly by taking the shared secret as is or indirectly by using it to derive the key. An *authenticated* key exchange further extends this notion by validating the identity and "aliveness" of both participants in the protocol, thus ensuring security even in the presence of an active adversary who attempts to participate in the protocol themselves.

For example, the Transport Layer Security (TLS) protocol, a cryptographic protocol designed to secure communications over a network, performs an authenticated key exchange at the start of the protocol [1]. The TLS protocol uses digital certificates to perform server and client (optional) authentication. This protocol is the de-facto standard for securing Web traffic and authenticating their source.

However, the current TLS standard does not protect the secure key exchange and authentication stages against a capable quantum computer. On July 2, 2022, the National Institute of Standards and Technology (NIST) announced the first winners of its Post-Quantum (PQ) Cryptography competition (PQC). NIST selected digital signatures Dilithium [2] and FALCON [3], and the Kyber key exchange for standardization. Of these, Commercial National Security Algorithm Suite (CNSA) 2.0 [4] only allows Kyber and Dilithium for the US Government. Consequently, technologies are migrating from current algorithms (classical) to the quantum resilient ones approved by NIST.

While there are proposals and Internet Engineering Task Force (IETF) Internet-Drafts (I-D) for incorporating PQ algorithms into TLS, moving to PQ will significantly increase the size of digital signatures and certificates. In some cases, PQ algorithms increase the signature size more than 40 times (e.g., moving from ECDSA-384 with signature of 96 bytes to Dilithium5 with $4,500$ bytes). As a result, PQ TLS is not suitable for devices in bandwidth-limited environments.

To overcome those limitations, we have developed the quantum resistant Compact Authenticated Key Exchange Hiding Identities (CAKE-HI) protocol. It creates a shared secret between two devices in an austere, dynamic, multi-system environment. The purpose of CAKE-HI is to be integrated into and provide shared secrets to other[1] communication protocols that may generate from them symmetric keys to protect the link and its traffic. CAKE-HI is designed to minimize computational and protocol overhead, while providing mutual authentication, forward secrecy, and

*This material is based upon work supported by the Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Dept. of the Air Force.*

*© 2026 Massachusetts Institute of Technology.*

1. CAKE-HI is not intended to replace or substitute data protection protocols like IPsec and/or TLS — but to complement them. For example, work is under way to integrate CAKE-HI into IKEv2.

quantum resilience. This protocol follows the generic idea of using static long-term verified or certified public keys in the key derivation process to accomplish implicit authentication, as defined in the MQV [5] and HMQV [6] papers. It is modeled after the approach [7] proposed for TLS, as described by P. Schwabe, et al. and reflected in their IETF KEMTLS[8] proposal.

The proposed protocol would benefit any use case, because it exchanges and processes fewer bits with equal results. But its advantages over the traditional "explicit signature" approach manifest best and become necessary in the following cases:

- Bandwidth-limited applications that include lossy narrow-band connections, where sending more bits across could increase the time required for completing handshake to beyond acceptable, or even make successful completion impossible. Error-caused retransmissions would exacerbate this further, as would the risk of prolonged exposure.
- Power-limited, such as battery-operated devices — e.g., attritable UAVs, where transmitting and/or processing extra bits would drain battery and eat into the device's operating time.

## 2. Background

### 2.1. Cryptography

**Plaintext and Ciphertext** or $pt$ and $ct$ are unencrypted and encrypted data respectively.

**Symmetric cryptography** is characterized by using one key, $k$ for encryption and decryption. As mentioned earlier, this key is often a random bit-string. This form of cryptography is useful for encrypting/decrypting data between parties that already have a shared secret. Either the key is a *pre-shared* key established beforehand, or it is established at the beginning of communication via a key exchange protocol. The most common symmetric encryption algorithm is AES [9].

**Public-key cryptography** or asymmetric cryptography uses a key pair with public and secret keys, ($pk$, $sk$). This class of cryptography has two main uses: digital signatures and key establishment. This allows cryptographic operations such as key agreement and authentication without shared secrets between parties. Intuitively, Alice may use Bob's public key, $pk^{bob}$, and Bob must use his private key, $sk^{bob}$ to decrypt the message. In other words, any party with another party's public key can encrypt a message, but only the party with the associated private key may decrypt the message. Thus, the confidentiality of messages hinges on keeping the private key $sk$ secret.

**Digital signatures** (the other main use of public key cryptography), play a critical role in authentication. Digital signatures support two operations: a signer can create a signature for a message using a secret key and any verifier with the public key and message can check the validity of the signature. When the public key is tied to an identity (e.g., via a certificate), we can assume that only that identity could have produced the signature for a given message.

**Digital Certificates** or certificates are digital files used to prove the validity of a public key. The certificate consists of the public key, metadata such as the name of the public key's owner, etc., and a digital signature over the public key and metadata. The certificate is signed by a trusted entity that vouches for the validity of the certificate.

**Key Encapsulation Mechanism** (KEM) is a method for deriving shared secrets with public key cryptography. Alice uses Bob's public key to generate a shared secret and a ciphertext that Bob can decode using his secret key to compute the same shared secret.

**Key Derivation Function** or KDF is an algorithm that derives one or more keys from a master key. The derived keys can be used in encryption algorithms.

**Hash functions** are one-way functions $H(msg)$ that map arbitrary-length messages to a fixed length (e.g., 384-bit) digest, and possess certain cryptographic properties. For example, given a digest, it is difficult to recover the $msg$ (preimage) or any other message that hashes to the same digest (second preimage). In the random oracle model, the output of the hash function is assumed to be uniformly distributed over the output space.

**Hashed Message Authentication Code** (HMAC) is an algorithm to ascertain the integrity and authenticity of a message with a hash function. A naive HMAC has the general structure of $f = H(k||m)$ where $H$ is a hash function, $k$ is a symmetric key, and $m$ is the message itself.

### 2.2. Quantum threat

In the mid 90's, Peter Shor developed Shor's Algorithm[10], which uses properties of a quantum computer to either factor large numbers or compute discrete logs in polynomial time. Nearly all Public Key Cryptography (PKC) used today relies either on the hardness of factoring large composite numbers (e.g., RSA) or computing discrete logs (e.g., DH, DSA, ECDH, ECDSA), and hence are vulnerable to Shor's Algorithm.

Though quantum computers exist today, none have enough processing capability to run Shor's Algorithm even for small primes. Despite that, the possibility that a powerful enough quantum computer may exist some day poses a threat to current cryptographic schemes. In fact, the threat of a Crypto-Relevant Quantum Computer (CRQC) applies immediately to the confidentiality of traffic (for the data whose lifetime is sufficiently long). This is due to the risk of intercepting and recording encrypted communications today and breaking their encryption if or when a CRQC becomes available ("record now, break later" attack). On the other hand, peer authentication during the key exchange and link establishment is not vulnerable until a CRQC becomes

available. Waiting to transition to QR PKC until a CRQC existence is publicly known means risking vulnerability during the transition period while waiting for migration of legacy devices to QR PKC.

### 2.3. Identity hiding

Often it is desirable for the communicating peers to conceal their identities from an outsider. It is infeasible to achieve it against an active adversary, as the identity of the peer that sends her credentials first would leak. Modern protocols usually take measures to conceal peers' identities from a passive adversary (an eavesdropper).

### 2.4. Relation to other protocols

The most prominent communication security protocols used today are TLS (in its multiple variants, such as DTLS, mTLS [11], cTLS [12], QUIC) and IPsec [13]. All create a secure communication channel for user data via a key exchange protocol. But there are two key differences between TLS, IPsec, and CAKE-HI.

First, the key exchange process for IPsec and TLS includes negotiating a set of options such as the algorithm suite and cryptographically signing the handshake transcript. CAKE-HI on the other hand minimizes its communication and computation overhead by using a fixed algorithm suite and not generating a signature over the handshake transcript. Second, this protocol only authenticates the communicating peers and establishes a shared secret. It does not secure the subsequent user data traffic and does not aim at replacing either TLS or IPsec. In other words, the actual employment of the generated shared secret to protect user traffic is considered out of scope. Its purpose is limited to providing an authenticated shared secret to other protocols, such as IPsec [14], MACsec [15], SCIP [16], etc.

KEMTLS [17] would offer benefits comparable to this proposal, however at the cost of (a) still being more bandwidth-demanding and computationally-expensive, and (b) bringing the TLS traffic protection overhead, which may be unacceptable for protocols like SCIP that already have their own traffic protection mechanisms, and only require secure keying.

## 3. CAKE-HI

Compact Authenticated Key Exchange Hiding Identities (CAKE-HI) is a quantum resistant key exchange protocol that minimizes the amount of communication to establish an authenticated, secure link. It minimizes the communication overhead by minimizing the use of signatures. In a bandwidth-limited environment, minimizing the communication overhead increases the likelihood of a successful protocol run. Eliminating signatures is impossible since there are no other practical ways to perform secure authentication. But, CAKE-HI avoids signature *generation* — it only uses an offline-generated certificate signature. Not only is signature generation expensive, in the case of PQ signature algorithms the signatures are very large. Instead, only one signature *verification* is performed by each of the communicating parties in order to authenticate the peer.

While both parties must share their certificates or identities for authentication, we assume the identities of each party shall remain confidential. They encrypt their certificates with a shared, ephemeral key that they generate via a KEM. This key is used to encrypt the certificate and is an input to the KDF when generating the shared key.

Instead of generating a signature over the handshake transcript like TLS, CAKE-HI performs an implicit authentication of the handshake. It does this by making the protocol's session key dependent on not only a series of KEM calculated shared secrets but also dependent on the hashes of the sent and received messages. Since only the corresponding party who they have authenticated can know those secrets, deriving the same session key implicitly authenticates each other when creating a shared secret.

### 3.1. Constraints

This protocol is geared toward low-bandwidth links that limit the size of physical packets. We understand that a protocol message larger than this limit would require fragmentation. It is undesirable, but may be unavoidable if public keys, ciphertexts, and signatures for PQ algorithms must be sent over the air. Thus, if a message's size is greater than the size of the physical packet, it will take more than one actual transmission to get the entire message across. The receiving entity pauses its execution of the protocol until the entire message arrives.

Since the protocol is fairly symmetric for both nodes, peers exchange messages asynchronously. Doing so allows nodes to perform the protocol in parallel. Lost messages are out of scope for the protocol.

### 3.2. Assumptions

The protocol functionality and security rely on the following assumptions. First, the initiator and responder have been provisioned with the appropriate Certificate Authority (CA) and their own certificate. In fact, it may make more sense for a larger network to employ multiple CAs and intermediate CAs such that not a single CA is trusted by the entire network. Each node can carry its own list of CAs and a policy that guides their use. If two nodes do not have a set of CAs in common, they will not be able to establish a secure link as they will not be able to verify each other's certificate.

Then, we assume strong sources of randomness for ephemeral key generation (KEM.Keygen) and seeds for the shared secrets $ss_*$ (in KEM.Enc()).

Lastly, we assume the use of a cryptographically strong hash function (e.g., SHA384), KDF (e.g., HKDF), KEM (e.g., Kyber or ML-KEM), and certificate signature algorithms (e.g., Dilithium5 or ML-DSA).

### 3.3. Notation

- $(sk_e, pk_e)$ — randomly-generated *ephemeral* key pair;
- $ss_{\{e,i,r\}}$ — randomly-generated shared secret (256-bit string) that is encapsulated in the ephemeral ($pk_e$), Initiator static ($pk_i$), or Responder static ($pk_r$) public key;
- $ss_{pre}$ — an optional pre-shared symmetric secret, usually network- or community-wide, to mitigate identity disclosure by an active attacker outside the network or community. An octet string when used, an empty value otherwise ;
- $ct_{\{e,i,r\}}$ — ciphertext for the ephemeral ($pk_e$), Initiator static ($pk_i$), or Responder static ($pk_r$) exchanges;
- $(ss, ct) \leftarrow \mathsf{KEM.Enc}(pk)$ — derive new random shared 256-bit key $ss$ and encapsulate into ciphertext $ct$ using public key $pk$;
- $ss \leftarrow \mathsf{KEM.Dec}(sk, ct)$ — decapsulate and derive 256-bit shared key $ss$ from ciphertext $ct$ using private key $sk$.
- KDF — Key Derivation Function, such as HKDF;
- $k_{C_{\{i,r\}}}$ — shared symmetric keys for Key Confirmation (256 bits each);
- $k_{session}$ — shared symmetric 256-bit keys generated by KDF from concatenation of decapsulated shared secrets (256 bits each);
- $C_{\{i,r\}}$ — HMAC values (384 bits) of the complete exchange transcript, computed using the appropriate key ($k_{C_{\{i,r\}}}$);
- $a||b$ — concatenation of two byte strings $a$ and $b$;

### 3.4. Protocol

The protocol is partitioned into four logical stages: establishing a confidential link and exchanging certificates, encapsulating and sending shared secrets, decapsulating shared secrets and deriving session keys, and key confirmation of the shared session key. The protocol diagram is shown in figure 1.

The numbers on the left side of the protocol diagram depict steps that the Initiator takes sequentially. Where a message *arrow* points towards an entity, that entity has to wait for reception of a message from its peer to continue.

**3.4.1. Establish link.** To begin the protocol, the initiator and responder create a secure (though unauthenticated) channel with each other, to protect the privacy of subsequent traffic, including their certificates. The initiator generates an ephemeral key pair $(sk_e, pk_e)$ and sends the public key $pk_e$ to the responder. A single ephemeral key exchange is sufficient: if the responder were to symmetrically initiate a second ephemeral key exchange, it would incur additional communication overhead without improving security. On receipt of the public key, the responder encapsulates the public key, creating a shared secret and ciphertext: $ss_e$, $ct_e$. The responder transmits the ciphertext and the initiator decapsulates the ciphertext to get the shared secret, $ss_e$. Both parties derive a symmetric key, $k_{hid}$, via the shared secret and (optional) $ss_{pre}$, which they use to encrypt and then transmit their respective certificates.

**3.4.2. Encapsulation.** The second stage of the protocol involves validating the signature of the received certificate and generating two more shared secrets. Each node uses $k_{hid}$ to decrypt the neighbor's encrypted certificate and validates the decrypted certificate using a trust anchor (such as a trusted Certificate Authority). If the validation fails, the node aborts the protocol. If successful, it extract the neighbor's static public key, $pk_{\{i,r\}}$ from the certificate. Both parties then generate shared secrets, $ss_{\{i,r\}}$, via the KEM's encapsulate function and exchange the resulting ciphertexts, $ct_{\{i,r\}}$.

**3.4.3. Decapsulation.** The third step in the protocol is deriving a session key via the set of shared secrets and the hash transcripts of the messages sent and received. Each node use the KEM to decapsulate a shared secret $ss_{\{i,r\}}$ from its secret key $sk_{\{i,r\}}$ and the ciphertext from its neighbor, $ct_{\{i,r\}}$. The set of shared secrets $ss_{\{e,i,r\}}$ and message transcript hashes are fed through a KDF to derive three keys: two key confirmation keys, $k_{C_{\{i,r\}}}$, and one session key, $k_{session}$. The key confirmation keys are used in the last step of the protocol to confirm both parties have derived the same session key without directly comparing the session key. CAKE-HI outputs the session key, $k_{session}$, which can be used to derive future shared secrets as necessary to secure the link.

**3.4.4. Key Confirmation.** Lastly, both parties must confirm they share the same session key before encrypting application data. To do so, both parties generate a key confirmation value, defined as an HMAC of the complete exchange transcript, computed using the appropriate key ($k_{C_{\{i,r\}}}$). They transmit their HMAC value, $C_{\{i,r\}}$, and compute their neighbor's HMAC value. Both parties should have derived the same set of confirmation keys and hashes of the protocol transcript. If their received HMAC value equals their computed value, the protocol is successful. If not, the authentication protocol failed and the node should immediately delete the derived keys and abort the protocol.

## 4. Informal Security Goals

We list here the infomal security goals of the CAKE-HI protocol and establish intuition for the method by which they can be proven.

1) **The secure channel between the Initiator and Responder is mutually authenticated**: the protocol halts if either $cert_i$ or $cert_r$ certificate fails validation. Otherwise, the certified public key $pk_{\{i,r\}}$ is used in the protocol. The key exchange ensures that each party must use its static secret key $sk_{\{i,r\}}$ to derive the correct shared secret $ss_{\{i,r\}}$ (e.g., in line 15 of figure 1, from which the shared key $S$ is derived.
2) **Key freshness**: If at least one of the participants follows this protocol – in particular, uses proper randomness to generate corresponding $ss_e$, $ss_i$, $ss_r$ – then the

CAKE-HI – Compact Authenticated Key Exchange Hiding Identities

| **Initiator** | | **Responder** |
|---|---|---|
| | ..... Establish confidential link and exchange certificates ..... | |
| 1 : | $(sk_e, pk_e) \leftarrow \mathsf{KEM.Keygen}()$ | |
| 2 : | $h^i_{ir} \leftarrow \mathsf{Sha384}(pk_e)$ | |
| 3 : | $\xrightarrow{\{pk_e\}}$ | $h^r_{ir} \leftarrow \mathsf{Sha384}(pk_e)$ |
| | | $(ss_e, ct_e) \leftarrow \mathsf{KEM.Enc}(pk_e)$ |
| | | $h^r_{ri} \leftarrow \mathsf{Sha384}(ct_e)$ |
| 4 : | $h^i_{ri} \leftarrow \mathsf{Sha384}(ct_e)$ | $\xleftarrow{\{ct_e\}}$ |
| 5 : | $ss_e \leftarrow \mathsf{KEM.Dec}(sk_e, ct_e)$ | |
| 6 : | $\mathsf{k}_{hid} \leftarrow \mathsf{KDF}(ss_e \vert\vert ss_{pre} \vert\vert \texttt{"HID"})$ | $\mathsf{k}_{hid} \leftarrow \mathsf{KDF}(ss_e \vert\vert ss_{pre} \vert\vert \texttt{"HID"})$ |
| 7 : | $h^i_{ir} \leftarrow \mathsf{Sha384}(h^i_{ir} \vert\vert \mathsf{E}_{k_{hid}}(cert_i))$ | $h^r_{ri} \leftarrow \mathsf{Sha384}(h^r_{ri} \vert\vert \mathsf{E}_{k_{hid}}(cert_r))$ |
| 8 : | $\xrightarrow{\{\mathsf{E}_{k_{hid}}(cert_i)\}}$ | $\xleftarrow{\{\mathsf{E}_{k_{hid}}(cert_r)\}}$ |
| | ..... Encapsulate and send shared secrets ..... | |
| 9 : | *Validate* $cert_r$ | *Validate* $cert_i$ |
| 10 : | $h^i_{ri} \leftarrow \mathsf{Sha384}(h^i_{ri} \vert\vert \mathsf{E}_{k_{hid}}(cert_r))$ | $h^r_{ir} \leftarrow \mathsf{Sha384}(h^r_{ir} \vert\vert \mathsf{E}_{k_{hid}}(cert_i))$ |
| 11 : | $(ss_i, ct_i) \leftarrow \mathsf{KEM.Enc}(pk_r)$ | $(ss_r, ct_r) \leftarrow \mathsf{KEM.Enc}(pk_i)$ |
| 12 : | $h^i_{ir} \leftarrow \mathsf{Sha384}(h^i_{ir} \vert\vert ct_i)$ | $h^r_{ri} \leftarrow \mathsf{Sha384}(h^r_{ri} \vert\vert ct_r)$ |
| 13 : | $\xrightarrow{\{ct_i\}}$ | $\xleftarrow{\{ct_r\}}$ |
| | ..... Decapsulate shared secrets and derive session keys ..... | |
| 14 : | $h^i_{ri} \leftarrow \mathsf{Sha384}(h^i_{ri} \vert\vert ct_r)$ | $h^r_{ir} \leftarrow \mathsf{Sha384}(h^r_{ir} \vert\vert ct_i)$ |
| 15 : | $ss_r \leftarrow \mathsf{KEM.Dec}(sk_i, ct_r)$ | $ss_i \leftarrow \mathsf{KEM.Dec}(sk_r, ct_i)$ |
| 16 : | $S \leftarrow ss_e \vert\vert ss_i \vert\vert ss_r \vert\vert ss_{pre} \vert\vert h^i_{ir} \vert\vert h^i_{ri}$ | $S \leftarrow ss_e \vert\vert ss_i \vert\vert ss_r \vert\vert ss_{pre} \vert\vert h^r_{ir} \vert\vert h^r_{ri}$ |
| 17 : | $k_{C_i}, k_{C_r}, \mathsf{k}_{session} \leftarrow \mathsf{KDF}(S)$ | $k_{C_i}, k_{C_r}, \mathsf{k}_{session} \leftarrow \mathsf{KDF}(S)$ |
| | ..... Key confirmation ..... | |
| 18 : | $\mathsf{C}_i \leftarrow \mathsf{HMAC}(k_{C_i}, h^i_{ir} \vert\vert h^i_{ri})$ | $\mathsf{C}_r \leftarrow \mathsf{HMAC}(k_{C_r}, h^r_{ri} \vert\vert h^r_{ir})$ |
| 19 : | $\xrightarrow{\{\mathsf{C}_i\}}$ | $\xleftarrow{\{\mathsf{C}_r\}}$ |
| 20 : | $\mathsf{C}_r \stackrel{?}{=} \mathsf{HMAC}(k_{C_r}, h^i_{ri} \vert\vert h^i_{ir})$ | $\mathsf{C}_i \stackrel{?}{=} \mathsf{HMAC}(k_{C_i}, h^r_{ir} \vert\vert h^r_{ri})$ |

Figure 1: CAKE-HI: a quantum resilient key exchange for bandwidth limited devices

shared keys generated by the protocol are unique and never used earlier (except for with a negligibly small probability). In particular, this implies that replaying old traffic messages will fail the integrity (MAC) check.

3) **At the end of the protocol, both the Initiator and Responder compute the same shared keys $k_{C_i}, k_{C_r}, k_{session}$ that are known only to them**. A passive adversary cannot learn $ss_e$ and hence not the shared session key. An active adversary attempting to corrupt messages cannot successfully complete the key confirmation step (i.e., the protocol does not end properly).
4) **Perfect Forward Secrecy**: Confidentiality of stored message data from future adversaries, even when the adversaries are given the long-term keys (i.e., $sk_i$ and/or $sk_r$). Perfect Forward Secrecy assumes that secrecy holds against adversaries without the secret keys (e.g., as in property 1). For CAKE-HI, this means that $k_{session}$ must be protected from the future compromise of the long-term keys. CAKE-HI achieves this through $ss_e$ from the ephemeral key exchange: even if an adversary uses the long-term keys to decapsulate $ss_i$ and $ss_r$ and the message hashes, it would require $ss_e$ to obtain $k_{session}$ from the KDF.
5) **Identity hiding**: The first ephemeral KEM produces a shared secret $ss_e$, and the protocol derives symmetric key $k_{hid}$ from it (and from $ss_{pre}$ if that option is used). Peers encrypt their certificates with it before exchanging them, thus hiding their identities from passive adversaries. If the optional $ss_{pre}$ is employed, active attackers "outside of the valid community" would be unable to decrypt received certificates.

# 5. Modelling CAKE-HI

The protocol is modeled in Verifpal [18] which verifies security properties in the symbolic model and in CryptoVerif [19] which verifies security properties by applying game transformations using the computational assumptions of cryptographic primitives.

## 5.1. Motivation

The use of mechanised provers for verification of protocols minimizes the risk of errors in security proofs, where mistakes could mean devastating attacks on cryptographic protocols. It is difficult to reason with respect to concurrent executions of protocols with modifications to messages made by an efficient adversary due to the large number of possibilities. In addition, formal specifications of protocols are unambiguous which makes necessary security assumptions clear and can allow for code generation if the serialization of the protocol messages is known.

## 5.2. Execution Environment

There are two honest parties in the model, the initiator and the responder, where the initator sends the first message in the protocol. We assume that the long term public keys of the initiator and responder have already been exchanged and can not be changed by the adversary. We prove security properties both for the initiator and the responder, since their actions are not perfectly symmetrical. We allow the adversary to represent multiple adversaries by generating multiple key pairs, to modify messages, and to execute the protocol multiple times and concurrently. We allow the adversary to call any random oracles or collision resistant hash functions that we use, and for some properties allow the adversary to obtain the long term secret keys or to determine the ephemeral key pair generated by the initiator. We model certificates as public keys, and do not model any additional metadata or signatures. We also perform the hashing of the protocol transcript all during the key derivation step, rather than immediately after receiving a message, but this does not affect the security proofs since it is not visible to the adversary.

## 5.3. CryptoVerif Model

We represent the protocol execution of a party as a sequence of message oracles, where the correct input to the oracle is the received message from the other party in the protocol and the output of the oracle is the computed output message from a protocol execution. Adversaries are represented as black-box interactive machines interacting with oracles. This includes probabilistic Turing machines or quantum adversaries which only have classical access to random oracles and whose runtime is bounded by a polynomial in various security parameters. We allow the adversary to access the message oracles sequentially, as in a real protocol execution. In parallel with the message oracles, the adversary can access a random oracle or collision resistant hash function being used in the protocol. For forward secrecy, the adversary is also allowed to obtain long-term secret keys in parallel with the message oracles. We represent the pre-exchange of public keys by generating key pairs during the setup and adding them to a table which is accessible by both parties. The adversary is allowed to know the public keys of the initiator and responder unless the security property tested is identity hiding, where two key pairs are generated and the public keys are given to the adversary but the key pair being used is chosen uniformly at random from one of the two key pairs.

## 5.4. Security Parameters

We define two security parameters $NI$ and $NR$ to represent the number of executions of the protocol for the initiator and responder. For forward secrecy, we define a security parameter $N$ to represent the number of executions of an oracle which allows for leaking the secret keys of either party and allows for executing the protocol with either the initiator or responder. We also define security parameters for the probability of breaking the properties of cryptographic primitives, the probability of certain kinds of collisions, and for the number of queries to the random oracle.

### 5.5. CryptoVerif Types

We use two assumptions on CryptoVerif types, boundedness and largeness. Boundedness requires a type to be finite, and largeness requires a type to be large enough such that collisions between elements from uniformly random sampling can be eliminated. In general, probabilistic Turing machines may only be able to approximate a uniform distribution for arbitrary finite sets [20], but this is sufficient for our security proofs.

We assume that the key and seed for the encryption algorithm, the seed and shared secret for the KEM, the index to the hash function family, and the output space of the PRF and key derivation function can be efficiently sampled from a close to uniform distribution. We also assume that the keyspace for the encryption algorithm and the output space of the PRF and key derivation function is large enough so that collisions with random elements can be eliminated.

### 5.6. Domain Separation

We assume generally that the domains of any random oracles are separated. To help separate the domains of the key derivation functions, we use a constant value which is added to the end of the input to one key derivation function but not the other. This value is known to the adversary.

### 5.7. CryptoVerif Definitions

The definitions of security properties in CryptoVerif are game-based definitions. We model each security property both with and without pre-shared keys. We give simplified definitions here, where full definitions are available in the security proofs.

1) **Mutual Authentication**
We define a mutually authenticated channel in CryptoVerif as a relation between events in the protocol. The events $authReqI, authReqR$ occur after the decapsulation of $ct_i, ct_r$ using the long term secret key, and the events $authEndI, authEndR$ occur if key confirmation is successful. The definition states that every $authEndI$ event is preceded by an $authReqR$ event, and that every $authEndR$ event is preceded by an $authReqI$ event with overwhelming probability. The addition of the values $ss_i$ and $ss_r$ in the query forces both events to occur in the same protocol execution with overwhelming probability, since at least one of $ss_i$ and $ss_r$ are sampled uniformly at random for each party and each protocol execution.

2) **Freshness of** $k_{C_i}, k_{C_r}, k_{session}$
We define a fresh key in round $i$ as one that is different from the keys generated in previous rounds. In CryptoVerif, we check for freshness by constructing a table with generated keys and checking if an adversary can cause the newly generated key to be equivalent to a previously generated key with non-negligible probability.

3) **Secrecy of** $k_{session}$
We define secrecy using the real-or-random game-based security definition which in CryptoVerif is defined using a secrecy query. Informally, secrecy means that the attacker can distinguish the secret from a fresh random value only with negligible probability. We also allow the adversary to determine the ephemeral key pair created by the initiator.

4) **Forward Secrecy**
We define forward secrecy using the same real-or-random game-based definition encoded in CryptoVerif, but where the protocol allows the adversary to obtain either both the long-term secret keys but not modify $pk_e$ and $ct_e$ or obtain the long-term secret key for the initiator but not modify $pk_e$.

5) **Identity Hiding**
We define identity hiding as computational indistinguishability between the outputs of the parties for any two valid key pairs $(sk_1, pk_1), (sk'_1, pk'_1)$ where the lengths of the public keys are the same $(Z(pk_1) = Z(pk'_1))$. We split the proof of identity hiding into a model for each party in the protocol to more easily build and debug proofs interactively. We also assume that the preshared secret $ss_{pre}$ is secret with respect to the adversary for identity hiding. For all other security properties, we allow the adversary to determine $ss_{pre}$.

6) **IK-CCA2**
We encode a new game-based security definition for the IK-CCA2 security property in CryptoVerif's language to prove the identity hiding property. This cryptographic assumption intuitively states that the encapsulated output of the KEM does not reveal information about the public key used to generate it.
We represent the probability of breaking the IK-CCA2 property in time $t$ for $N_k$ key generations, $N_{Encap}$ encapsulations, and $N_{Decap}$ decapsulations by an efficient adversary as $P_{key}$. We assert that the adversary's advantage in distinguishing between two cryptographic games is probability $P_{key}$, where the main difference between the games is that in the second game the public key used for the encapsulation is generated from a uniformly random seed, and the decapsulation is performed using a lookup table.

7) **Implicitly Rejecting KEM**
We encode a new game-based security definition for a implicitly rejecting KEM based on CryptoVerif's current definition of a KEM. The only difference is that the return type is a bitstring rather than a bitstringbot and there is no use of the injective function injbot to

map from the secret key to bitstringbot.

## 5.8. Cryptographic Assumptions

We require that all of the properties here hold against quantum adversaries, but may not mention it explicitly.

1) **Key Derivation Function**
   We model the key derivation function in the protocol using a random oracle hash function for the second use of the key derivation function in the protocol. For the forward secrecy proofs, we model the first use of the key derivation function as a random oracle, otherwise we use a PRF. We assume that the inputs to the key derivation function are associated with the correct arguments and that the domains of the key derivation functions are appropriately separated. This security assumption is justified since HKDF is based on HMAC, which is proven to have quantum security in the random oracle model [21].

2) **IND-CCA2 Security of $E_{k_{hid}}$**
   We require the output of the encryption algorithm to be indistinguishable under adaptive chosen ciphertext attacks. This is justified since AES-GCM is an authenticated encryption algorithm, which is known to have stronger security properties than IND-CCA2.

3) **IND-CCA2 and IK-CCA2 Security of Implicitly Rejecting KEM**
   We require the KEM to provide anonymity [22] along with the output being indistinguishable under adaptive chosen ciphertext attacks. We also require the KEM to be implicitly rejecting, meaning that it does not abort if the input to the decryption algorithm is not a valid ciphertext. This is justified since Kyber is proven to be SPR-CCA secure which implies that it is both IND-CCA2 secure and IK-CCA2 secure [23]. It is also an implicitly rejecting KEM [24].

4) **PRF assumption of HMAC**
   We require HMAC to be a pseudorandom function, which is satisfied if the compression function is a pseudorandom function [25].

5) **Collision resistance of Sha384**
   We assume that Sha384 is a collision resistant hash function for the freshness security property. This property is justified in [26], where the sponge construction is proven to be indifferentiable from a random oracle assuming that the permutation used is a random permutation.

## 5.9. Proof Guidance

We explain here the intuition behind the manual proof guidance provided to CryptoVerif.

**5.9.1. Freshness, Secrecy, and Forward Secrecy.** For freshness, secrecy, and variants of forward secrecy, the first step is to apply the IND-CCA2 property of a KEM used for ($ss_e$ or $ss_i, ss_r$), which we can use since the public key used for encapsulation is restricted to be a valid public key of the initiator or responder. This causes the output shared secret to be indistinguishable from a random value of the same output length and allows us to apply the random oracle assumption, resulting in keys which are fresh and indistinguishable from random.

**5.9.2. Authentication.** We perform the same steps to prove authentication as for secrecy but also apply the PRF assumption of HMAC, which results in a key confirmation value which is indistinguishable from a value output by a random function since the key used is secret. This means that completing key confirmation implies that the decapsulation operation of the KEM was used with the correct secret key.

**5.9.3. Identity Hiding.** The first step is to apply the IND-CCA2 assumption of the encryption algorithm, which causes the output public key to be indistinguishable from another public key of the same length. The next step is to apply the IK-CCA2 assumption, which causes the public keys used to generate the encapsulations to be indistinguishable from other valid public keys. The remaining possibilities to obtain information about the key pair used are from the key confirmation value and whether the key confirmation received is correct.

Cases are added by CryptoVerif to split the possible outcomes of the key confirmation oracle and key confirmation values, where the cases are based on whether any messages have been modified by the adversary and whether the adversary queried the initiator oracle or the responder oracle first. The indistinguishability of the key pairs given these oracles is confirmed in CryptoVerif by reducing these cases and showing that the branches for each key pair are equivalent.

## 5.10. Verifpal Models

Verifpal is not proven to be sound or complete in the symbolic model, since it does not in general check all possible computations of the attacker (it terminates after a max depth or until a fixpoint is reached). However, there are existing translations to the protocol analysis framework ProVerif for authentication and confidentiality queries and to the theorem prover Rocq for confidentiality queries which both have formalized semantics and proofs of soundness.

## 5.11. Pre-Exchange of Public Keys

We represent the pre-exchange of public keys between the initiator and responder using guarded constants, which are not modifiable but are knowable to the attacker. For the identity hiding proof, the public keys are not revealed to the attacker but are assumed to be already known by both parties.

### 5.12. Verifpal Security Definitions

We use an active attacker in all of the Verifpal models and only use pre-shared keys in the identity hiding model for Verifpal. We also assume that the parties in the Verifpal models already have knowledge of what public keys are valid rather than verifying a signature from a certificate authority, where the set of valid public keys modeled is the key pairs of the two parties. We do not assume that the nodes will use out of band information to check that the decrypted certificates match the identity of the other party, since that will lead to the attack in section 6.3. The assert primitive is meant to be interpreted as a check for membership in the set of valid certificates, rather than checking for equality for a specific certificate.

1) **Mutual Authentication**
   To check for mutual authentication in Verifpal, we used authentication queries to check that a key confirmation value not generated by either party can not pass the key confirmation check. It is possible for the required key confirmation value to be generated by the same party which is checking the key confirmation value: this results in a form of self-authentication. We successfully submitted a patch to change the definition of authentication to only include cases where dishonest parties are able to generate a value which causes checked primitives using the constant in the query to pass.

2) **Freshness and Secrecy of** $k_{C_i}, k_{C_r}, k_{session}$
   We check for freshness and secrecy using the security definitions encoded into Verifpal. The definition of confidentiality is defined as the inability of an attacker to compute a term in the symbolic model given the constants that it can learn by modifiying protocol executions, and the definition of freshness is based on whether the attacker can force a value to stay the same and used successfully between sessions.

3) **Forward Secrecy**
   We use Verifpal's support for phases and leakage to model forward secrecy in Verifpal. The model for forward secrecy leaks the secret key after the exchange of $ct_i$ and $ct_r$ but before key confirmation, and a phase is declared before the secret keys are leaked to prevent values in previous phases from being modified using the knowledge of the secret keys.
4) **Identity Hiding**
   For identity hiding in Verifpal, we prove the confidentiality of each party's certificates. We do not use the definition of unlinkability in Verifpal, since the definition only requires that the unlinkable values be fresh and that they can not be determined from the same primitive, which does not match our infomal definition of identity hiding.

### 5.13. Cryptographic Assumptions

We are able to prove the security properties corresponding to the definitions in Verifpal using the following assumptions:

1) **Key Derivation Function**
   We model both of the key derivation functions used in the protocol using the HKDF primitive.

2) **Symmetric Key Encryption** $E_{k_{hid}}$**,** $D_{k_{hid}}$
   We model the encryption algorithm as the ENC/DEC primitives.

3) **Public Key Encryption** $KEM.Enc$**,** $KEM.Dec$
   We model the KEM as the PKE_ENC/PKE_DEC primitives, with the public key as the exponentiation of a group element and the secret key as the exponent.

4) **MAC assumption for HMAC**
   We model HMAC as the MAC primitive and its verification using the ASSERT primitive.

5) **Collision resistance of Sha384**
   We model the hashing of the protocol transcript using the HASH primitive along with CONCAT.

## 6. Verification Results

We use the terms $h^{i,r}_{sent}$ and $h^{i,r}_{recv}$ to mean the computed hashes of the messages sent and received by a given party.

### 6.1. Attacks

During the verification of the protocol, we obtained attacks on variants of the protocol with weaker assumptions. We also modify the computation of the protocol transcript to make the identity hiding property provable in CryptoVerif.

### 6.2. Reflection Attack

If only one confirmation key is used in the protocol, we can use the symmetry of the protocol to get the initiator to authenticate with itself. If the attacker makes the hash of the both the protocol transcripts $h^i_{sent}||h^i_{recv}$ and $h^i_{recv}||h^i_{sent}$ the same by sending the messages sent by the initiator back to the initiator, then $C_i = HMAC(k_C, h^i_{sent}||h^i_{recv}) = HMAC(k_C, h^i_{recv}||h^i_{sent}) = C_r$, so the key confirmation values sent by the initiator and responder will be the same and the key confirmation check passes. The encrypted certificate will also pass verification, since after decryption the result will be the initiator's certificate, and we assume that the initiator's certificate has a valid signature.

### 6.3. Certificate Validation

If a node aborts due to an incorrect identity during certificate validation obtained out of band (outside of the

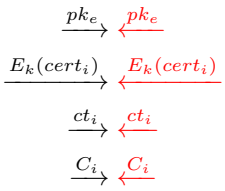


Figure 2: Reflection Attack

protocol), it can leak information about the identity of the other participant.

The attacker first changes $ct_e$ such that the decapsulation of $ct_e$ with $sk_e$ by the initiator results in $ss_e$. The encryption of the inititator's certificate $E_k(cert_i)$ and $ss_e$ are stored. When the attacker later observes a protocol execution, it can replace $ct_e$ with a value such that the decapsulation of $ct_e$ with $sk'_e$ results in the same value of $ss_e$ [27] (and the same value of $k$), and replace the encrypted certificate sent by the responder with $E_k(cert_i)$. If the node aborts, the attacker learns that the responder was not the initiator from the first execution, and if the node doesn't abort, the attacker learns that the responder was the initiator from the first execution.

To fix this, we require that the node only validates that the signature for the certificate is correct or only aborts based on out of band information at the end of the protocol. For similar reasons, we require the KEM to be implicitly rejecting.

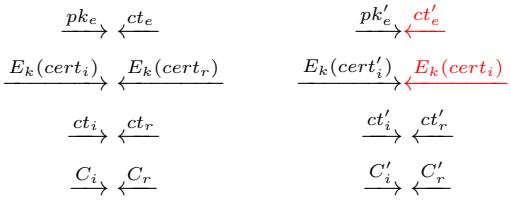


Figure 3: Certificate Validation Attack

## 6.4. HMAC Attack

Let the primitive used to compute the key confirmation value be a UF-CMA MAC instead of a PRF: it can now leak the hash of the transcript, since a MAC does not provide secrecy guarantees. Change the protocol to compute the protocol transcript with the hash of the decrypted certificate, rather than the encrypted certificate.

First, the attacker can observe a protocol execution and store the values of $ct_e$, $ct_r$, and $h^i_{recv}$. If the attacker wants to check during a different protocol execution if the initiator matches the responder from the first protocol execution, it can replace $ct_e$ and $ct_r$ with the stored value, replace $E_k(cert_r)$ with $E_k(cert'_i)$, and check if $h^i_{recv} = h^i_{recv'}$. If the hash values are equivalent, this implies that the decrypted certificates are equivalent, otherwise this implies that the decrypted certificates are not equivalent.

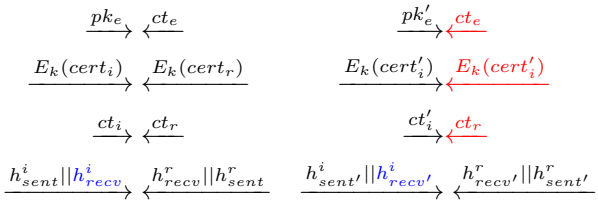


Figure 4: HMAC Attack

We fix this by requiring HMAC to be a PRF and by modifying the protocol to hash the encrypted certificate rather than the hashing the decrypted certificate. This prevents the hash of the protocol transcript from possibly leaking information about the public keys of the parties.

## 6.5. Summary

We prove mutual authentication, freshness and secrecy of session keys, forward secrecy, and identity hiding in CryptoVerif and Verifpal and define the required assumptions and security properties of cryptographic primitives. We also obtain attacks on variants of the protocol, modify the protocol in order to prove a stronger identity hiding property, and encode a new game-based definition of key indistinguishability and implicitly rejecting KEM into CryptoVerif.

## 6.6. Future Work

**6.6.1. Soundness.** While CryptoVerif and Verifpal do not have formal proofs of soundness, CryptoVerif presents informal proofs, a formalization of their semantics, and a translation to F*, and Verifpal performs translations to ProVerif and Rocq. To achieve stronger soundness properties, either a formal proof of soundness could be provided for CryptoVerif or the protocol properties could be proven directly in a theorem prover such as Rocq or F*.

**6.6.2. Stronger Adversary Model.** CryptoVerif is unable to prove some properties in the quantum random oracle model, where quantum adversaries can access random oracles in superposition [28]. We are unable to easily replace the use of the random oracle for the second key derivation function using primitives defined in CryptoVerif. If $ss_{pre}$ is available, it can be used as the key for a pseudo-random function that can replace the random oracle. However, this would only remove the random oracle assumption for the proof of anonymity, and the key is not completely secret since it is known among members of the community of interest.

Despite this challenge, we believe that the proofs should carry over in the QROM since we do not use proof techniques such as rewinding. It may be possible to replace the use of the random oracle in the protocol with a computational extractor, or a proof system can be used which can prove security in the quantum random oracle model such as [29].

**6.6.3. Stronger Security Properties.** To allow proofs of security to compose well with other security proofs, the proofs could be updated to use simulation based security definitions or security definitions from the universal composability framework [30], which are supported by [31]. This would allow CAKE-HI to be more easily integrated into other protocols with proofs of security.

## 7. Protocol Performance

In the case of bandwidth limited devices, the time required to complete the protocol is dominated by its over-the-air communications overhead. The overhead is a function of the number of messages each protocol stage needs to exchange, latency and throughput of the link, and the size of the KEM public keys and digital signature.

Due to their larger key and signature size, PQ algorithms will significantly increase the size of any key exchange protocol. For example, classical mTLS requires 672 bytes for a pair of Elliptic Curve Diffie-Hellman public keys, Elliptic Curve Digital Signature Algorithm (ECDSA) public keys, and four ECDSA signatures. A full classical mTLS key exchange amounts to the same amount of bytes as a FALCON-1024 public key. Unfortunately, this is the trade-off for PQC. But, CAKE-HI reduces the PQC overhead by eliminating the need for transmitting a live signature for each party.

We performed a performance analysis of CAKE-HI and mTLS 1.3 with PQ support via Open Quantum Safe's PQ TLS implementation [32]. Both protocols used Dilithium5 signatures to authenticate their peer and Kyber-1024 as the key encapsulation mechanism. FALCON-1024 is another PQ signature scheme and is the most compact scheme, but is not included in the CNSA 2.0 algorithm suite. We include FALCON1024 and a Dilihtium5 CA signed node certificate in our analysis to demonstrate the reduction in size and performance a more compact scheme can bring.

To compare both protocols, we collected the number of bytes they must send over-the-air and their cryptographic computations. Comparing their cryptographic computation requirements rather than their runtime has the benefit of being implementation agnostic and also establishes a theoretical lower-bound on their performance.

### 7.1. Handshake Size

CAKE-HI significantly reduces the amount of data needed to establish a PQ link over prominent alternatives such as mTLS. The breakdown of the difference in protocol handshake size for CAKE-HI and PQ TLS are given by Table 4. Since the main difference between the two protocols is the signature size, the table highlights the effect of the choice of signature scheme for CAKE-HI and mTLS as well as CAKE-HI's improvement over mTLS.

By being a signature-less protocol, CAKE-HI requires 8,586 fewer bytes than mTLS with FALCON-1024 and Dilithium5 signed certificates and public keys. This translates to a 44.2% decrease in handshake size for the Dilithium5 case and 67.2% for FALCON-1024. Using FALCON-1024 for both signature generation and verification in mTLS mirrors CAKE-HI's handshake size for a total of 13,190 bytes. If PQ security is required but CNSA 2.0 algorithms are not, the FALCON-1024 algorithm reduces the size to 34.1% and 23.7% for CAKE-HI and mTLS respectively. The reduction in size should reduce the time to establish a PQ secure link as it will require less data to be transmitted and/or retransmitted in the case of a noisy link.

It should be noted that PQ certificates are an area of active research and have not been finalized. As a result, we estimate a PQ certificate for CAKE-HI will be around 6,563 bytes. We arrived at 6,563 by summing the following: a Kyber-1024 public key (1568), Dilithium5 signature (4595), and metadata (400). This number may change once the structure of PQ certificates are finalized. However, the largest contributors to the certificate size are the public key and the digital signature itself, which in all likelihood will not drift far, if at all, from its current size. Therefore, our expectation is that the final certificate size would vary little, if at all, from the number we estimated.

Regardless of the format, CAKE-HI does not transmit a signature during the handshake like TLS does. Therefore, CAKE-HI should always outperform mTLS in this category, but by exactly how much remains to be seen.

### 7.2. Cryptographic Computations

Computational efficiency of either protocol will not noticeably impact their run-time performance as over-the-air communications are the dominating factor. However, efficiency gains for small battery-powered unmanned platforms such as drones can have a significant impact on run-time. For example, efficient algorithms may increase the flight time of drones. The comparison between both protocols is given in Table 3.

CAKE-HI requires one more KEM encapsulation and decapsulation for each party, while mTLS requires more rounds of hashing and a signature generation for each party. The hashing and signature generation are considerable given that mTLS requires 474.9% more rounds of hashing and one signature generation for each party. In the case of Dilithium5, signature generation is 56.4% more expensive than signature verification. Since the number of hash rounds is directly proportional to the handshake size, CAKE-HI requires fewer rounds. Thus, CAKE-HI is a more efficient alternative to mTLS.

## 8. Summary and future work

We described a minimalistic authenticated key exchange CAKE-HI designed to make use of post-quantum cryptographic algorithms possible in bandwidth-limited environments. It accomplishes this by (a) performing only those operations that are necessary to provide required cryptographic properties, and (b) replacing active digital signatures with implicit authentication. Comparison with alternatives such as TLS or IKEv2 are favorable to CAKE-HI.

Our next steps will be: collecting empirical measurements of CAKE-HI incorporated into the actual protocols under varying assumptions and bandwidth constraints to benchmark it against the current explicit authentication approach, and bringing CAKE-HI to the standards bodies, in hope to have it incorporated into protocols such as IKEv2.

## 9. Acknowledgments

This project is funded by the USAF. The authors want to thank Richard Davis (NSA), Adam Margetts (MIT Lincoln Laboratory) and Stephen Timmel (NSA) for reviewing this work and giving helpful suggestions.

## References

[1] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, Aug. 2018. DOI: 10.17487/RFC8446 [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[2] L. Ducas et al., "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 1, pp. 238–268, Feb. 2018. DOI: 10.13154/tches.v2018.i1.238-268 [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/839

[3] P.-A. Fouque et al., "FALCON: Fast-fourier lattice-based compact signatures over NTRU," 2019.

[4] *Announcing the commercial national security algorithm suite 2.0*, Sep. 2022. [Online]. Available: https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF

[5] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An efficient protocol for authenticated key agreement," *Designs, Codes and Cryptography*, vol. 28, no. 2, pp. 119–134, Mar. 2003.

[6] H. Krawczyk, "Hmqv: A high-performance secure diffie-hellman protocol," in *Advances in Cryptology – CRYPTO 2005*, V. Shoup, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 546–566, ISBN: 978-3-540-31870-5.

[7] P. Schwabe, D. Stebila, and T. Wiggers, "Post-quantum TLS without handshake signatures," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1461–1480, ISBN: 9781450370899. DOI: 10.1145/3372297.3423350 [Online]. Available: https://doi.org/10.1145/3372297.3423350

[8] S. Celi and T. Wiggers, *"post-quantum tls without signatures"*, Jan. 2021. [Online]. Available: https://blog.cloudflare.com/kemtls-post-quantum-tls-without-signatures/

[9] M. Dworkin et al., *Advanced encryption standard (AES)*, en, Nov. 2001. DOI: https://doi.org/10.6028/NIST.FIPS.197

[10] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700

[11] M. Badra and I. Hajjeh, "MTLS: (D)TLS Multiplexing," Internet Engineering Task Force, Internet-Draft draft-badra-hajjeh-mtls-06, Apr. 2011, Work in Progress, 16 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-badra-hajjeh-mtls/06/

[12] E. Rescorla, R. Barnes, H. Tschofenig, and B. M. Schwartz, "Compact TLS 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-ctls-08, Mar. 2023, Work in Progress, 24 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-ctls/08/

[13] K. Seo and S. Kent, *Security Architecture for the Internet Protocol*, RFC 4301, Dec. 2005. DOI: 10.17487/RFC4301 [Online]. Available: https://www.rfc-editor.org/info/rfc4301

[14] C. Kaufman, *Internet Key Exchange (IKEv2) Protocol*, RFC 4306, Dec. 2005. DOI: 10.17487/RFC4306 [Online]. Available: https://www.rfc-editor.org/info/rfc4306

[15] "Ieee standard for local and metropolitan area networks–media access control (mac) security," IEEE Std 802.1AE, 2018.

[16] *Secure communications interoperable protocol*. [Online]. Available: https://en.wikipedia.org/wiki/Secure_Communications_Interoperability_Protocol

[17] T. Wiggers, S. Celi, P. Schwabe, D. Stebila, and N. Sullivan, "KEM-based Authentication for TLS 1.3," Internet Engineering Task Force, Internet-Draft draft-celi-wiggers-tls-authkem-03, Apr. 2024, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-03

[18] N. Kobeissi, G. Nicolas, and M. Tiwari, *Verifpal: Cryptographic protocol analysis for the real world*, Cryptology ePrint Archive, Paper 2019/971, https://eprint.iacr.org/2019/971, 2019. [Online]. Available: https://eprint.iacr.org/2019/971

[19] B. Blanchet, "A computationally sound automatic prover for cryptographic protocols," in *Workshop on the link between formal and computational models*, Paris, France, Jun. 2005.

[20] L. Klingler, R. Steinwandt, and D. Unruh, "On using probabilistic turing machines to model participants in cryptographic protocols," *Theoretical Computer Science*, vol. 501, pp. 49–51, 2013, ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2013.05.015 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397513003666

[21] A. Hosoyamada and T. Iwata, *On tight quantum security of HMAC and NMAC in the quantum random oracle model*, Cryptology ePrint Archive, Paper

2021/774, 2021. [Online]. Available: https://eprint.iacr.org/2021/774

[22] T. Saito, K. Xagawa, and T. Yamakawa, "Tightly-secure key-encapsulation mechanism in the quantum random oracle model," in *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part III 37*, Springer, 2018, pp. 520–551.

[23] V. Maram and K. Xagawa, "Post-quantum anonymity of kyber," in *IACR International Conference on Public-Key Cryptography*, Springer, 2023, pp. 3–35.

[24] J. Bos et al., "Crystals-kyber: A cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2018, pp. 353–367.

[25] M. Bellare, "New proofs for nmac and hmac: Security without collision resistance," *Journal of Cryptology*, vol. 28, pp. 844–878, 2015.

[26] J. B. Almeida et al., "Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of sha-3," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 1607–1622, ISBN: 9781450367479. DOI: 10.1145/3319535.3363211 [Online]. Available: https://doi.org/10.1145/3319535.3363211

[27] C. Cremers, A. Dax, and N. Medinger, *Keeping up with the KEMs: Stronger security notions for KEMs and automated analysis of KEM-based protocols*, Cryptology ePrint Archive, Paper 2023/1933, 2023. [Online]. Available: https://eprint.iacr.org/2023/1933

[28] B. Blanchet and C. Jacomme, "Post-quantum sound CryptoVerif and verification of hybrid TLS and SSH key-exchanges," in *Proceedings of the 37th IEEE Computer Security Foundations Symposium (CSF'24)*, Enschede, The Netherlands: IEEE, Jul. 2024, pp. 543–556.

[29] M. Barbosa et al., *EasyPQC: Verifying post-quantum cryptography*, Cryptology ePrint Archive, Paper 2021/1253, 2021. DOI: 10.1145/3460120.3484567 [Online]. Available: https://eprint.iacr.org/2021/1253

[30] R. Canetti, *Universally composable security: A new paradigm for cryptographic protocols*, Extended Abstract appeared in proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS), 2001. canetti@watson.ibm.com 13131 received 22 Dec 2000, last revised 13 Dec 2005, 2000. [Online]. Available: http://eprint.iacr.org/2000/067

[31] R. Canetti, A. Stoughton, and M. Varia, *EasyUC: Using EasyCrypt to mechanize proofs of universally composable security*, Cryptology ePrint Archive, Paper 2019/582, 2019. [Online]. Available: https://eprint.iacr.org/2019/582

[32] D. Stebila and M. Mosca, "Post-quantum key exchange for the Internet and the Open Quantum Safe project," in *Selected Areas in Cryptography (SAC) 2016, LNCS*, R. Avanzi and H. Heys, Eds., vol. 10532, Oct. 2017, pp. 1–24.

# Appendix

## 1. Proof Metrics

We peform benchmarking of the verification time for Verifpal on an Intel i9-13950HX and measure the proof sizes for the pre-shared key variants of the security properties in CryptoVerif. The proofs in CryptoVerif complete in less than one minute on average.

To test properties in Verifpal such as secrecy, freshness, and forward secrecy which are combined together, we removed queries or parts of the proof which were not used for a specific security property to measure verification times.

| Security Property | Proof Size |
|---|---|
| Secrecy | 78 games |
| Forward Secrecy Variant 1 | 35 games |
| Forward Secrecy Variant 2 | 80 games |
| Authentication | 215 games |
| Freshness | 106 games |
| Identity Hiding (Initiator) | 333 games |
| Identity Hiding (Responder) | 307 games |

TABLE 1: Proof Metrics for CryptoVerif

| Security Property | Verification Time |
|---|---|
| Secrecy | $\approx$ 15 hours |
| Forward Secrecy | $\approx$ 5 minutes |
| Authentication | $\approx$ 10 hours |
| Freshness | $\approx$ 12 minutes |
| Identity Hiding | $\approx$ 40 minutes |

TABLE 2: Proof Metrics for Verifpal

## 2. Protocol Comparison

| Operation | CAKE-HI Initiator | CAKE-HI Responder | mTLS Client | mTLS Server |
|---|---|---|---|---|
| *Ephemeral Key Exchange* | | | | |
| Kyber1024.Keygen | 1 | 0 | 1 | 0 |
| Kyber1024.Encap | 0 | 1 | 0 | 1 |
| Kyber1024.Decap | 1 | 0 | 1 | 0 |
| *Authentication* | | | | |
| Dilithium5.Sign | 0 | 0 | 1 | 1 |
| Dilithium5.Verify | 1 | 1 | 1 | 1 |
| Kyber1024.Encap | 1 | 1 | 0 | 0 |
| Kyber1024.Decap | 1 | 1 | 0 | 0 |
| *Certificate and Metadata Encryption* | | | | |
| AES256-GCM | 2 | 2 | 9 | 9 |
| *Hashes* | | | | |
| SHA384 calls | 12 | 12 | 45 | 45 |
| SHA384 rounds | 175 | 175 | 1006 | 1006 |

TABLE 3: CAKE-HI vs mTLS key exchange cryptographic computation comparison

| CA Signatures | Signature Size in Bytes | CAKE-HI Signature Count | mTLS Signature Count | CA Signature Bytes | CAKE Hanshake Bytes | TLS Handshake Bytes | Handshake Difference |
|---|---|---|---|---|---|---|---|
| Dilithium5 | 4,595 | 2 | 2 | 9,190 | 19,398 | 27,984 | 8,586 |
| FALCON1024 | 1,280 | 2 | 2 | 2,560 | 12,768 | 21,354 | 8,586 |
| Difference | 3,315 | 0 | 0 | 6,630 | 6,630 | 6,630 | |

TABLE 4: CAKE-HI vs mTLS key exchange with Kyber1024-Dilithium5